

GPS Antenna Controller

By Steven A. Bailey

**NASA
June 23, 1999**

Table of Contents

INTRODUCTION.....	3
DESIGN.....	4
SOURCE CODE.....	8
GPS1.ASM	8
TABLEGEN5.M.....	14
FLOWCHART	15

Introduction

The Scanning Radar Altimeter (SRA) is an airborne radar ranging device requiring GPS position to remove aircraft motion. Due to GPS signal loss while performing aircraft turns, three GPS receivers are flown with antennas pointing at zenith, and complementary angles off zenith to maintain GPS lock on at least one receiver. Obviously, this has lead to extra hardware and post processing to recover continuous GPS position.

It is the goal of the GPS antenna controller to reduce the hardware to one GPS receiver with an antenna mounted on a controllable swivel. The design we chose was simple. A GPS antenna is mounted in a custom assembly which pivots in one axis from -45 to +45 degrees from zenith. A heavy duty model airplane servo drives this assembly via a short swing arm. See Figure 1.

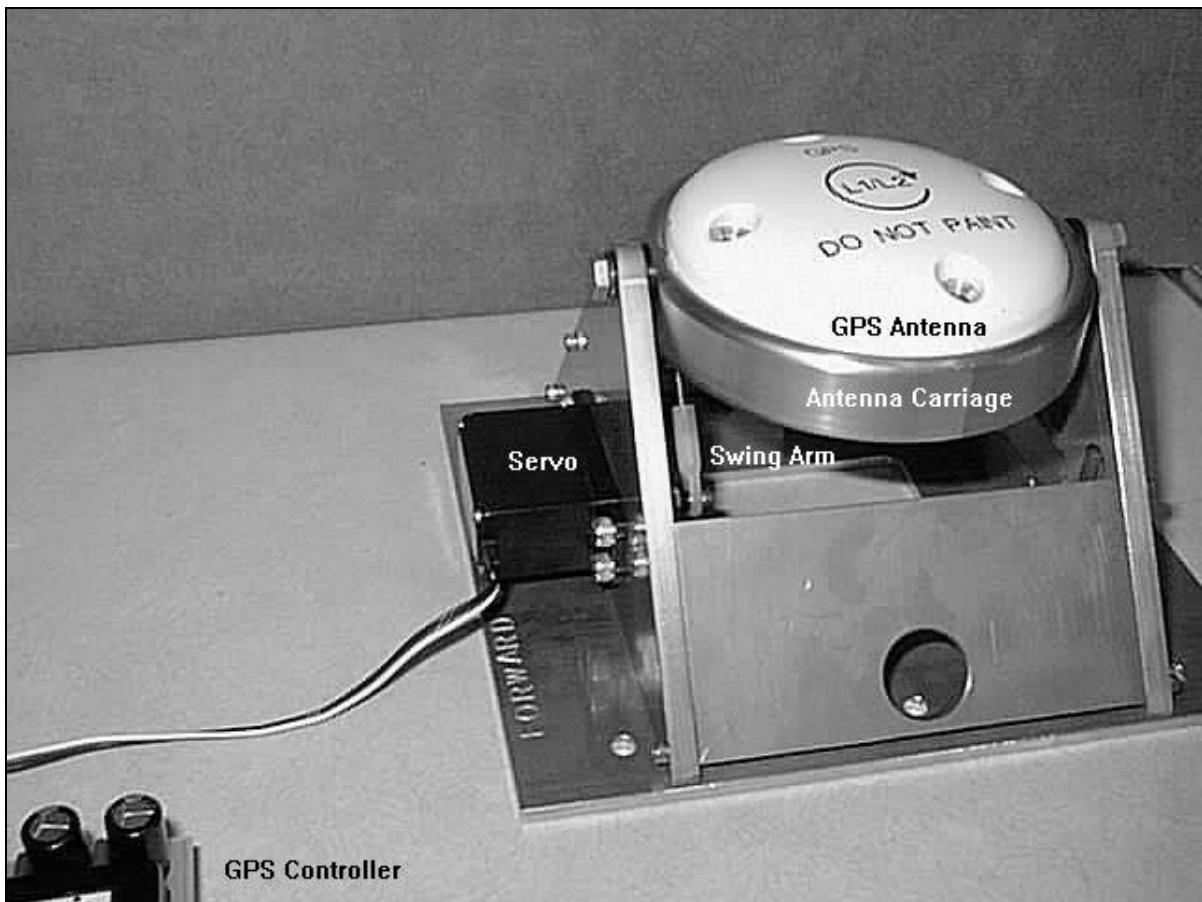


Figure 1

A custom antenna controller was designed, built, and tested in under 2 weeks using parts left over from past projects. Again, simplicity was the order. A microcontroller with onboard pulse width modulation (PWM) circuitry was used to drive the servo. The external interface

to the controller was RS-232. An external PC or terminal issues single byte commands requesting the antenna move to a given angle. The controllers job is then to position the servo to the said angle. See Figure 2.

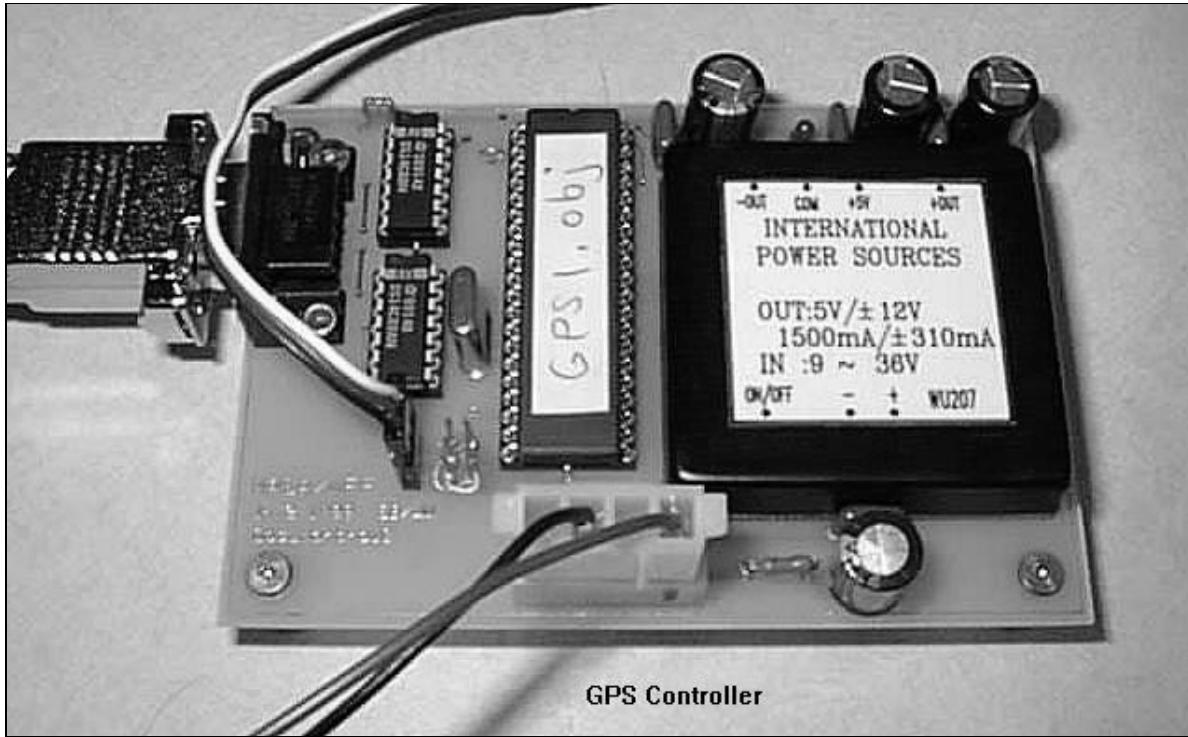


Figure 2

Design

There are only 4 major components to the GPS controller. A wide input range (9-36 volts) DC-DC converter (International Power Sources) is used to provide +5, +12, and -12 volts to the circuitry. A Microchip 16C65A microcontroller is used to provide PWM to the external servo as well as provide a communications link to a remote PC or terminal. Finally, one each RS-232 driver and receiver chip are used to provide appropriate signal levels to and from the controller. See Figure 3.

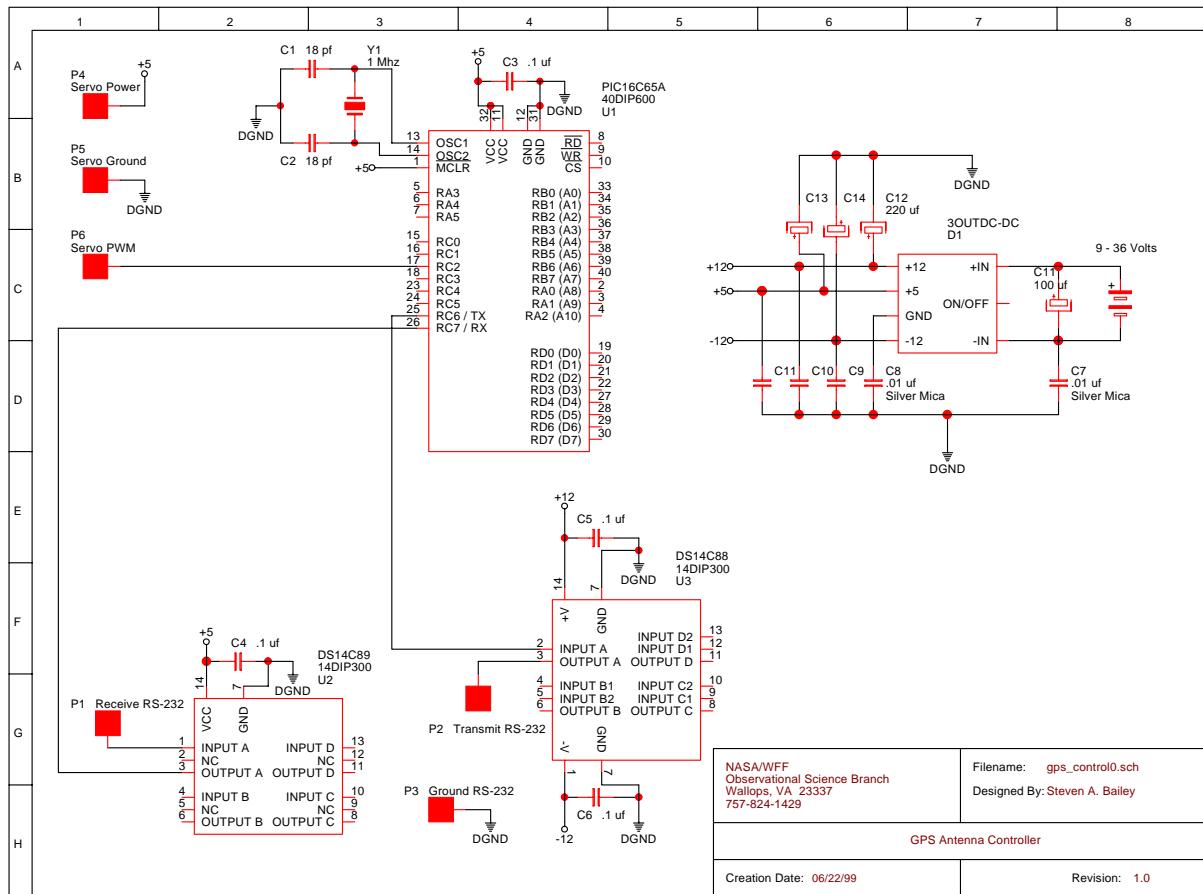
A machinist protractor was used to accurately determine antenna angle to 1 degree increments. The antenna swing arm connecting the servo to the antenna carriage was made long enough to provide a 2:1 ratio of servo angle to antenna angle. The idea was to run the servo through its full range (~180 degrees).

Also, using a pulse generator, I was able to determine the matching duty cycle necessary to achieve any given antenna angle. Taking measurements at -45, 0, and 45 degrees, I made a linear least squares fit and created Table 1.

The desired PWM frequency for most servos is 50 Hz. We chose 61 Hz because the math worked out to this while using a 1 Mhz crystal to drive the microcontroller. Higher frequencies can be used, but the duty cycle increases causing the servo to draw more current. Remember, these model servos expect a constant PWM no matter what the frequency. Therefore, as the frequency increases (with a constant PWM), the duty cycle increases.

From Table 1, we can see our controller angle resolution is very close to 1 degree. Actually, it is 1.12 degrees per PWM register value. This is close enough for our purposes.

One byte commands are issued from a connecting PC or terminal when an angle change is required. All commands are ascii and sequentially arranged. This way, a minimum computer with terminal software can easily talk to the controller. Since we are using a servo to the drive the antenna, our controller does not require a feedback loop to keep the servo on position. This is a feature of servos, they contain their own feedback loop.



Angle	PWM (usecs.)	PWM Register	Ascii Command
-45	900	56	
-44	914	57	!
-43	928	58	"
-42	942	59	#
-41	956	60	\$
-40	970	61	%
-39	984	62	&
-38	998	62	-
-37	1012	63	(
-36	1026	64)
-35	1040	65	*
-34	1055	66	+
-33	1069	67	'
-32	1083	68	-
-31	1097	69	.
-30	1111	69	/
-29	1125	70	0
-28	1139	71	1
-27	1153	72	2
-26	1167	73	3
-25	1182	74	4
-24	1196	75	5
-23	1210	76	6
-22	1224	76	7
-21	1238	77	8
-20	1252	78	9
-19	1266	79	:
-18	1280	80	;
-17	1294	81	<
-16	1308	82	=
-15	1322	83	>
-14	1337	84	?
-13	1351	84	@
-12	1365	85	A
-11	1379	86	B
-10	1393	87	C
-9	1407	88	D
-8	1421	89	E
-7	1435	90	F
-6	1449	91	G
-5	1464	91	H
-4	1478	92	I
-3	1492	93	J
-2	1506	94	K
-1	1520	95	L
0	1534	96	M
1	1548	97	N
2	1562	98	O
3	1576	99	P
4	1590	99	Q
5	1604	100	R
6	1619	101	S
7	1633	102	T
8	1647	103	U
9	1661	104	V
10	1675	105	W
11	1689	106	X
12	1703	106	Y
13	1717	107	Z
14	1731	108	[

15	1746	109	\
16	1760	110]
17	1774	111	^
18	1788	112	-`
19	1802	113	a
20	1816	114	b
21	1830	114	c
22	1844	115	d
23	1858	116	e
24	1872	117	f
25	1886	118	g
26	1901	119	h
27	1915	120	i
28	1929	121	j
29	1943	121	k
30	1957	122	l
31	1971	123	m
32	1985	124	n
33	1999	125	o
34	2013	126	p
35	2028	127	q
36	2042	128	r
37	2056	128	s
38	2070	129	t
39	2084	130	u
40	2098	131	v
41	2112	132	w
42	2126	133	x
43	2140	134	y
44	2154	135	Y
45	2168	136	z

Table 1

Source Code

Gps1.asm

The following source code was written in Microchip Assembler with heavy use of macros. A lookup table is used to set the PWM register accordingly given a one byte command. Remember, since we are using a 1 Mhz crystal, we must use the _XT_OSC configuration bit. The serial port baud rate is 1200 baud.

```
;-----  
; Gps1.asm  This is the first version of the GPS controller.  The  
;           goal is to control the azimuth of a small GPS antenna  
;           mounted on a swivel.  This swivel is driven by a model  
;           airplane servo.  This controller then provides a PWM  
;           input to the servo which causes rotation to desired  
;           angle.  
;  
;           Ascii commands are received by this controller.  Commands  
;           are ascii and 1 byte long. Values range from the SPACE char  
;           to the 'z' char.  There are 91 possible values corresponding  
;           to angles of -45 to 45 degrees.  
;  
;           ' ' = -45 degrees  
;           'M' = 0 degrees  
;           'z' = +45 degrees  
;  
;-----  
; Steven A. Bailey      NASA      6/18/99  
;-----  
  
list          p=16c65a, r=dec    ; Define processor and 'dec'  
              ; numbers as default  
  
include <p16c65a.inc>          ; Processor specific defines  
include <pxmacs.inc>          ; Parallax 'like' macros  
  
              ; These are the configuration  
              ; bits found in 'p16c65a.inc'.  
  
__config     _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF & _BODEN_ON  
  
;-----  
; Program defines  
;-----  
  
#define     SPBRG_VAL    12      ; Baud Rate = low  
              ; SPBRG_VAL = (FOSC - BAUDRATE*64)  
              ; -----  
              ; (BAUDRATE*64)  
              ;  
              ; For 1.2 Kbaud using 1 Mhz FOSC:  
              ;  
              ; SPBRG_VAL = (1,000,000 - 1200*64)  
              ; -----  
              ; (1200*64)  
              ;  
              ; = 12
```

```

;
;*****
;
#define PR2_VAL      255 ; Freq =           1
;-----_
;      (PR2_VAL + 1)(4*(1/FOSC)*PRE_VAL)
;
; For FOSC = 1 Mhz and PRE_VAL = 16
;
; Freq =           1
;-----_
;      (255 + 1)(4 * (1/1,000,000) * 16)
;
;      = 61 Hz
;
;*****
;

#define PWM      PORTC,2 ; PWM output

;-----
; Variables in RAM
;-----
; A more consistent way to define RAM is to use the 'org' statement
; followed by variables using the 'res' directive. However, the RAM
; window of 'MPLAM' does not display variables defined this way...hence
; we are using the 'equ' directive followed by the address in the
; register map they are located.
;-----
;          org  0x20      ; Start of RAM in 16c65a
;
;var1      res  1      ; Examples using 'res'
;var2      res  1
;-----

pwm       equ   0x20      ; PWM variable
rcv_byte  equ   0x21      ; Receive RS-232 variable
xmt_byte  equ   0x22      ; Transmit RS-232 variable
temp      equ   0x23      ; Temp variable

;-----
; On startup, the PIC looks at address 0 for its first
; instruction. Since the interrupt handler begins at
; address 4, we'll just jump over it to get to the
; startup routine.
;-----
;          org  0      ; Base vector location
;          jmp  start    ; Beginning of main program.

;-----
; Here's the startup routine and the main program loop.
;-----

start
    call    initialize     ; Initialize CPU
mainloop
    snb    PIR1,RCIF      ; Skip if Receiver not ready
    call    parse          ; Parse command from serial port
    jmp    mainloop
;
```

```

; sendchar to serial port
; From call to return, this function takes 8 cycles (best case) or
; 32 usec. Worst case dependent on baud rate.
;-----

sendchar
    sb      PIR1,TXIF          ; Skip if PIR1,TXIF bit set
    jmp     sendchar

    mov_ff  TXREG,xmt_byte    ; Now...transmit xmt_byte

    return                         ; Return to caller

;-----
; Initialize serial port and variables
; From call to return, this function takes 70 cycles or 280 usecs.
;-----

initialize
    setb   STATUS,RP0           ; Switch to Memory Bank 1

    mov_f1 TRISA,00000000b      ; Make ra0-7 outputs
    mov_f1 TRISB,00000000b      ; Make rb0-7 outputs
    mov_f1 TRISC,10000000b      ; Make rc7 pin input
    mov_f1 TRISD,00000000b      ; Make rd0-7 outputs
    mov_f1 TRISE,00000000b     ; Make re0-7 outputs

    mov_f1 SPBRG,SPBRG_VAL    ; Set baud rate generator
    mov_f1 TXSTA,00100000b      ; Trans enable, 8-bits, async, baud low
    mov_f1 PR2,PR2_VAL         ; Set PWM period

    clrb   STATUS,RP0           ; Switch to Memory Bank 0

    mov_f1 RCSTA,10010000b      ; Serial port enable
    mov_ff  rcv_byte,RCREG       ; Clear out receive buffer
    mov_f1 T2CON,00000110b      ; Turn TMR2 on with 16x prescale

    mov_f1 CCPR1L,00011000b      ; Set PWM to 0 degs (96 dec)
    mov_f1 CCP1CON,00001100b     ; Enable PWM

    return                         ; Return to caller

;-----
; Parse command from input from serial port
; From call to return, this function takes 215 cycles (worst case) or
; 860 usecs.
;
; This function takes a byte from the serial port and saves into
; 'rcv_byte'. This value is then checked for out-of-bounds values
; < 32 or > 122. An inbounds value has 32 subtracted off of it and
; the result is used as an offset into our jump table to retrieve
; the appropriate PWM.
;-----

parse
    mov_ff  rcv_byte,RCREG       ; Read byte from serial port
    cjb_f1 rcv_byte,32,endret    ; Ignore if rcv_byte < 32
    cja_f1 rcv_byte,122,endret    ; Ignore if rcv_type > 122

```

```

        mov_ff      xmt_byte,rcv_byte      ; Echo back char if in range
        call       sendchar

        sub_f1      rcv_byte,32          ; Subtract 32 from rcv_type

        mov_f1      PCLATH,HIGH table   ; Get high address of table
        mov_wf      w,rcv_byte         ; Load w with lower 8 bits
        call       table              ; Get next table entry
        mov_fw      pwm,w              ; Get pwm value back from table
        mov_f1      PCLATH,HIGH parse  ; Restore current high address

        mov_ff      temp,pwm           ; Save copy of pwm again
        rr          temp               ; Remove 2 LSBs
        rr          temp               ; Mask out 2 MSBs
        and_f1     temp,00111111b     ; Copy temp to CCPR1L
        mov_ff      CCPR1L,temp        ; Mask out other bits

        mov_ff      temp,pwm           ; Save copy of pwm in temp
        rl          temp               ; Get lower 2 bits in <5:4>
        rl          temp               ; position
        rl          temp               ; Enable PWM
        and_f1     temp,00110000b     ; Copy <5:4> bits into CCP1CON
        or_f1      temp,00001100b
        mov_ff      CCP1CON,temp

endret
        return

-----
; Table lookup. This table contains the value which must be 'stuffed'
; into the register combination CCPR1L & CCP1CON<5:4> to give the
; desired PWM. Remember that CCPR1L contains the 8 MSBs and
; CCP1CON<5:4> contains the 2 LSBs. With 1 Mhz crystal, prescaler
; of 16, and TMR2 prescale value of 255...should get 61 Hz PWM with
; resolution of 1.8 degrees using Airtronics servo with 1:1 hookup.
;
; From call to return, this function takes 6 cycles or 24 usecs.
-----

```

```

org    0x800

table
        add_fw      PCL,w

        retw      56      ; -45 Degrees
        retw      57      ; -44 Degrees
        retw      58      ; -43 Degrees
        retw      59      ; -42 Degrees
        retw      60      ; -41 Degrees
        retw      61      ; -40 Degrees
        retw      62      ; -39 Degrees
        retw      62      ; -38 Degrees
        retw      63      ; -37 Degrees
        retw      64      ; -36 Degrees
        retw      65      ; -35 Degrees
        retw      66      ; -34 Degrees
        retw      67      ; -33 Degrees
        retw      68      ; -32 Degrees
        retw      69      ; -31 Degrees
        retw      69      ; -30 Degrees
        retw      70      ; -29 Degrees

```

retw	71	; -28 Degrees
retw	72	; -27 Degrees
retw	73	; -26 Degrees
retw	74	; -25 Degrees
retw	75	; -24 Degrees
retw	76	; -23 Degrees
retw	76	; -22 Degrees
retw	77	; -21 Degrees
retw	78	; -20 Degrees
retw	79	; -19 Degrees
retw	80	; -18 Degrees
retw	81	; -17 Degrees
retw	82	; -16 Degrees
retw	83	; -15 Degrees
retw	84	; -14 Degrees
retw	84	; -13 Degrees
retw	85	; -12 Degrees
retw	86	; -11 Degrees
retw	87	; -10 Degrees
retw	88	; -9 Degrees
retw	89	; -8 Degrees
retw	90	; -7 Degrees
retw	91	; -6 Degrees
retw	91	; -5 Degrees
retw	92	; -4 Degrees
retw	93	; -3 Degrees
retw	94	; -2 Degrees
retw	95	; -1 Degrees
retw	96	; 0 Degrees
retw	97	; 1 Degrees
retw	98	; 2 Degrees
retw	99	; 3 Degrees
retw	99	; 4 Degrees
retw	100	; 5 Degrees
retw	101	; 6 Degrees
retw	102	; 7 Degrees
retw	103	; 8 Degrees
retw	104	; 9 Degrees
retw	105	; 10 Degrees
retw	106	; 11 Degrees
retw	106	; 12 Degrees
retw	107	; 13 Degrees
retw	108	; 14 Degrees
retw	109	; 15 Degrees
retw	110	; 16 Degrees
retw	111	; 17 Degrees
retw	112	; 18 Degrees
retw	113	; 19 Degrees
retw	114	; 20 Degrees
retw	114	; 21 Degrees
retw	115	; 22 Degrees
retw	116	; 23 Degrees
retw	117	; 24 Degrees
retw	118	; 25 Degrees
retw	119	; 26 Degrees
retw	120	; 27 Degrees
retw	121	; 28 Degrees
retw	121	; 29 Degrees
retw	122	; 30 Degrees
retw	123	; 31 Degrees
retw	124	; 32 Degrees

retw	125	; 33 Degrees
retw	126	; 34 Degrees
retw	127	; 35 Degrees
retw	128	; 36 Degrees
retw	128	; 37 Degrees
retw	129	; 38 Degrees
retw	130	; 39 Degrees
retw	131	; 40 Degrees
retw	132	; 41 Degrees
retw	133	; 42 Degrees
retw	134	; 43 Degrees
retw	135	; 44 Degrees
retw	136	; 45 Degrees

Tablegen5.m

The following is Matlab source code used to generate a PWM lookup table used in 'Gps1.asm'.

```
%-----  
% tablegen5.m...This Matlab program generates a table of  
% azimuth vs. PWM duty cycle (in usecs) vs.  
% CCPR1L:CCP1CON<5:4> bit values for 16c645a  
% PWM generation. A 1 Mhz clock is assumed  
% with a prescaler of 16 giving us a PWM  
% period of 61 Hz.  
%  
% Using an Futaba servo, I measured 3  
% angles (-45,0,45) with a machinist  
% protactor.  
%  
% -45 deg = 900  usec on pulse  
% 0 deg    = 1530 usec on pulse  
% +45 deg = 2172 usec on pulse  
%  
% using a 61 Hz. period  
%  
% A linear fit worked best and here are the  
% coefficients:  
%  
% m = 14.1  
% b = 1534  
%  
% To regenerate table, use the following  
% linear equation:  
%  
% PWM = 14.1*azimuth + 1534  
%-----  
% SAB      NASA      6/15/1999  
%-----  
  
x      = zeros(91,3);          % Create space  
x(:,1) = (-45:45)';          % Azimuth  
x(:,2) = x(:,1).*14.1 + 1534; % PWM  
x(:,3) = (x(:,2).*1e-6)./(1/1000000)*16; % CCPR1L...  
  
fid = fopen('table5.txt', 'w');  
fprintf(fid, '%3d  %5.0f  %4.0f\n', x);      % Write table  
fclose(fid);
```

Flowchart

